Master 1 Informatique et Ingénierie des Systèmes Complexes (IISC)

Université de Cergy-Pontoise

Atelier de gesion de projet

Rapport de projet : BDA

Auteurs :
Djahid ABDELMOUMENE
Amine AGRANE
Ishak AYAD
Abdelhakim SAID
Donald LAY
Yacine ZABAT

Sujet proposé par :
Mr. Tianxiao LIU
Mr. Dan VODISLAV

Rapport remis le 23 janvier 2020

Table des matières

Table des figures

1 Introduction à la partie B.D.A.

Le but de cet atelier, pour la partie B.D.A., est d'implémenter des fonctionnalités simples de recherche mettant en oeuvres des requêtes sur une base de données dites "étendue" (BDe) mêlant une partie relationnelle (SQL) et une autre textuelle gérée par le moteur d'indexation Lucene. Une API a été conçue pour l'exploitation des données en provenance de la BDe.

2 Modèle conceptuel des données

Ici, nous allons présenter notre modèle conceptuel des données. Après la lecture du sujet et la concertation entre les membres du groupe, nous sommes arrivés au modèle conceptuel de données suivant :

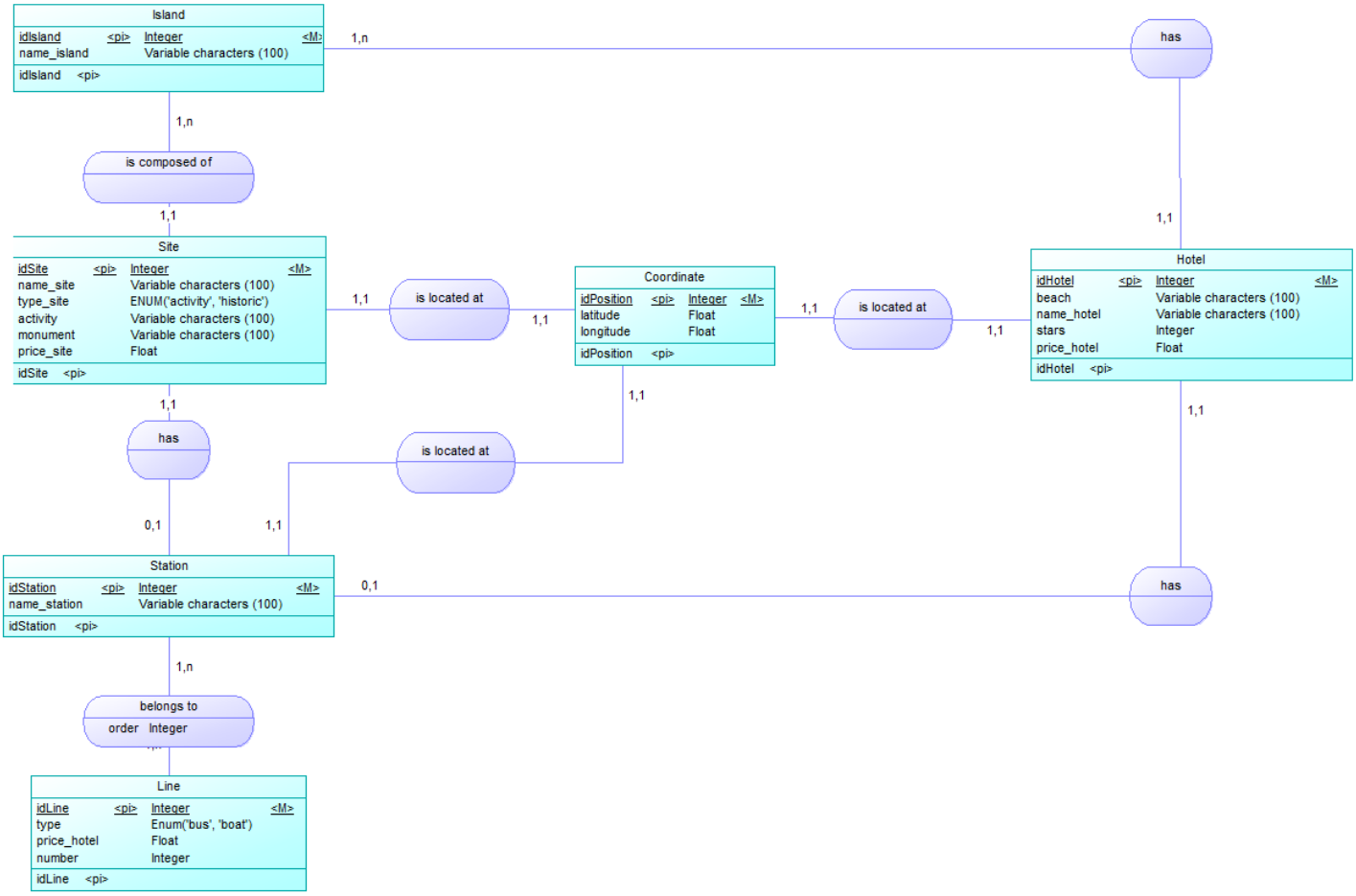

FIGURE 1 – Modèle conceptuel des données

Dans cette partie, nous discutions de la conception de notre base de données relationnelles. Après lecture du sujet, nous avons décidé d'utiliser 6 tables afin d'assurer au mieux la gestion des informations utilisées par notre système. Nous avons donc conçu une table "island", une table "hotel", une table "station", une table "site", une table "line", une table "coordinate", et enfin une table "s_belongs_to_l".

— La table "island" contient des informations sur les îles (identifiant unique et nom de l'île). L'identifiant de l'île se trouve aussi en tant que clé étrangère dans les tables "hotel" et

"site". C'est à dire qu'à partir de l'identifiant d'une île, on peut déterminer l'ensemble des hôtels et des sites touristiques présents sur cette île.“

— La table "coordinate" est utilisée afin de stocker un ensemble de coordonnées géographiques. Chaque coordonnée sera définie par son identifiant unique, ainsi que sa longitude et latitude.
— On utilise les table "line" pour enregistrer les lignes de transports présente dans notre base de données. Pour chaque ligne, on enregistre l'identifiant unique de la ligne, le numéro de la ligne (par exemple ligne numéro 14), le type de la ligne (ligne de bus ou ligne maritime) et enfin le prix de déplacement sur cette ligne.
— En ce qui concerne la table "station", on sauvegardera dans cette dernières les différentes stations qui composent les lignes de transports de notre base de données. Une station sera définie par son identifiant unique, son nom, ainsi qu'un clé étrangère vers un élément de la table "position", et ce afin de pouvoir déterminer la position de la station.
— Dans la table "s_belongs_to_l", on effectue le lien entre les éléments de la table "line" et les éléments de la table "station". Pour se faire, chaque ligne de la table "s_belongs_to_l" aura pour attribut un identifiant unique, un numéro spécifiant l'ordre de la station et une clé étrangère pour chacune des tables "line" et "station".
— On utilise la table "hôtel" pour sauvegarder l'ensemble des informations relatives aux hôtels présents dans notre base de données. Pour chaque hôtel, on stockera son identifiant unique, son nom, le nombre d'étoiles de l'hôtel, le nom de la plage incluse dans l'hôtel, le prix de l'hôtel. Pour chaque hôtel correspondra une unique île, station et position. On incorpore cette notion par l'ajout des clés étrangères dans notre table hôtel.
— Enfin, la table "site" ou on stocke l'ensemble des sites touristiques présents dans notre base de données. Un site touristique est constitué d'un identifiant unique, du nom du site, du type du site (activité ou monument historique), du prix du site. Pour chaque site touristique corresponds une unique île, station et position. On modélise cette relation par l'ajout de clés étrangères vers les tables "island", "station" et "coordinate".

3 Modèle logique des données

Dans cette section, nous offrons une nouvelle aproche pour visualiser la conception de notre base de données à l'aide du modèle logique de données :

island(id_island, name_island)
line(id_line, number_line, price, number)
coordinate(id_position, latitude, longitude)
station(id_station, name_station,*id_position)
s_belongs_to_l(id, station_order, *id_station, *id_line)
hotel(id_hotel, beach, name_hotel, stars, price_hotel, *id_island, *id_position, *id_station)
site(id_site, name_site, type_site, price_site, activity, monument, *id_island, *id_position, *id_station)

4 Conception de la partie base de données du système

Nous allons à présent discuter de la conception UML de notre partie "bde". C'est cette partie qui s'occupe de la gestion des données de notre application. Dans notre projet, la partie bde est divisé en quatre packages :

— **persistence/jdbc** : JDBC est une bibliothèque écrite en Java permettant d'exploiter des bases de données relationnelles. Dans notre projet, nous avons opté pour l'utilisation de JDBC au lieu d'Hibernate. Le package persistence/jdbc contient les classes qui se chargent d'établir la connexion avec notre base de données MySQL et assurent la bonne exécutions de nos requêtes sur la BDD.
— **lucene** : Le package Lucene contient les classes qui se chargent de la manipulation et des traitements sur les fichiers de descriptions. Il permet notamment la création des fichiers de descriptions, les création d'index, et la recherche textuelle sur nos descriptions.
— **api** : Contient la classe ApiFacade, cette dernière constitue une interface entre l'utilisateur et notre système.
— **iterator** : Le package "iterator" implémente le design pattern de même nom, il nous permet une élévation en abstraction afin de manipuler des objets itérables sans en connaître la structure. Il sera utilisé lors des traitements sur les objets itérables retournés par JDBC et Lucene.

5 Conception UML : API

Nous allons à présent évoquer notre API, son rôle est crucial, c'est en effet cette dernière qui se chargera d'effectuer les différentes taches demandées dans le projet. Elle implémente, entre autres, la gestion de trois types de requêtes :

1. Requête SQL : ce sont les requêtes à destination de la base de données relationnelle. Elles sont prises en charge par JDBC ;
2. Requête textuelle : ce sont les requêtes mettant en oeuvre l'index créé par Lucene ;
3. Requête mixte : ce sont des requêtes comportant une partie relationnelle et une autre textuelle. Elles sont gérée par notre propre opérateur de jointure implémenté dans la classe NestedLoopJoin.java.

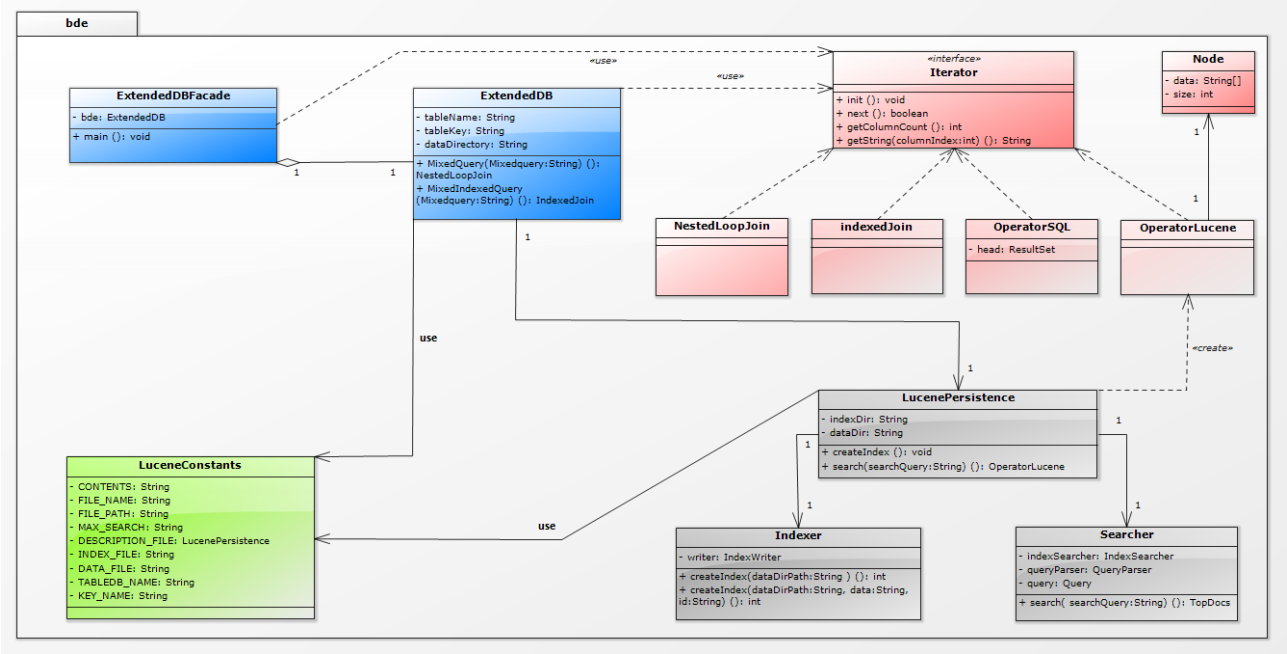

FIGURE 2 – Diagramme de classe pour la base de données étendu

5.1 Conception UML : JDBC

Le package JDBC contient des classes qui assurent la connexion et l'interaction avec notre base de données (l'exécution de requêtes SQL). Une fois que l'API effectue le parsing de la requête, elle envoie la partie SQL vers JDBC qui lui se charge d'exécution de la requête. Le package JDBC se compose de deux classes, la classe "JdbcConnection.java" et la classe "JdbcPersistance.java". La classe JdbcConnection se charge d'établir la connexion avec notre base de données MySQL. La classe JdbcPersistance quand à elle gère la communication avec la base de données : elle exécute les requêtes qu'on lui passe et récupère le résultat correspondant.

5.2 Conception UML : Lucene

Le package Lucene est implémenté via deux sous-package : core et persistence. Le package core contient trois classes : Indexer, LuceneBuilder, et Searcher.
LuceneBuilder : C'est la classe chargée de la création des fichiers textuels. Elle prends en argument le fichier description qu'elle lit, parse, et divise en fichiers textuels. Le nom de chaque fichier correspond à l'id d'un site touristique dans notre base de données, le contenu du fichier quant à lui correspond à la description textuelle du site touristique.
Indexer : Cette classe se charge de la création d'un index sur les fichiers textuels qui contiennent les descriptions de nos sites touristiques.
Searcher : La classe "Searcher" permet d'effectuer une recherche dans nos fichiers textuels à l'aide de l'index. Cette dernière utilise deux classes Lucene importées : IndexSearcher et QueryParser. Pendant que la première effectue une recherche dans l'index créé, QueryParser est responsable de la traduction de la requête de recherche en informations que la machine peut comprendre.
Le package persistence est constitué de deux classes : **LuceneConstants** et **LucenePersistence**. Dans **LuceneConstants**, on définit l'ensemble des constantes qui seront utilisées par les différentes méthodes présentes dans le package Lucene. En ce qui concerne la classe **LucenePersistence**, on utilise nos classes Searcher et Indexer afin d'implémenter des méthodes nous permettant de créer un index sur nos fichiers et de retourner le résultat d'une recherche sous forme d'un itérateur.

6 Plans d'exécution : étude comparative

La recherche par mots clés, qui constitue une des fonctionnalités de base attendues pour ce projet, est notamment assurée par la gestion de requêtes comportant une partie SQL et une autre textuelle. La recherche textuelle est assurée par le moteur d'indexation lucene.

Pour réaliser l'interprétation de ce genre de requête, il nous a été suggéré de réaliser un arbre de calcul où les feuilles constitueraient les termes (représentant les résultats des requêtes SQL et textuelle sous la forme d'objets itérables) et leur racine, le résultat de l'opération de jointure entre ces deux objets (également sous la forme d'un objet itérable).

Nous ont alors été proposés deux plans d'exécution pour l'implémention de l'opérateur joignant les résultats SQL et textuel. Ces deux plans suivent la même logique, à savoir, pour une itération du membre gauche (Iterator A), on itère entièrement le membre de droite (Iterator B). Ainsi, pour les deux plans d'exécution, si on a n et m les nombres respectifs de résultats pour les requêtes A et B, nous aurons :

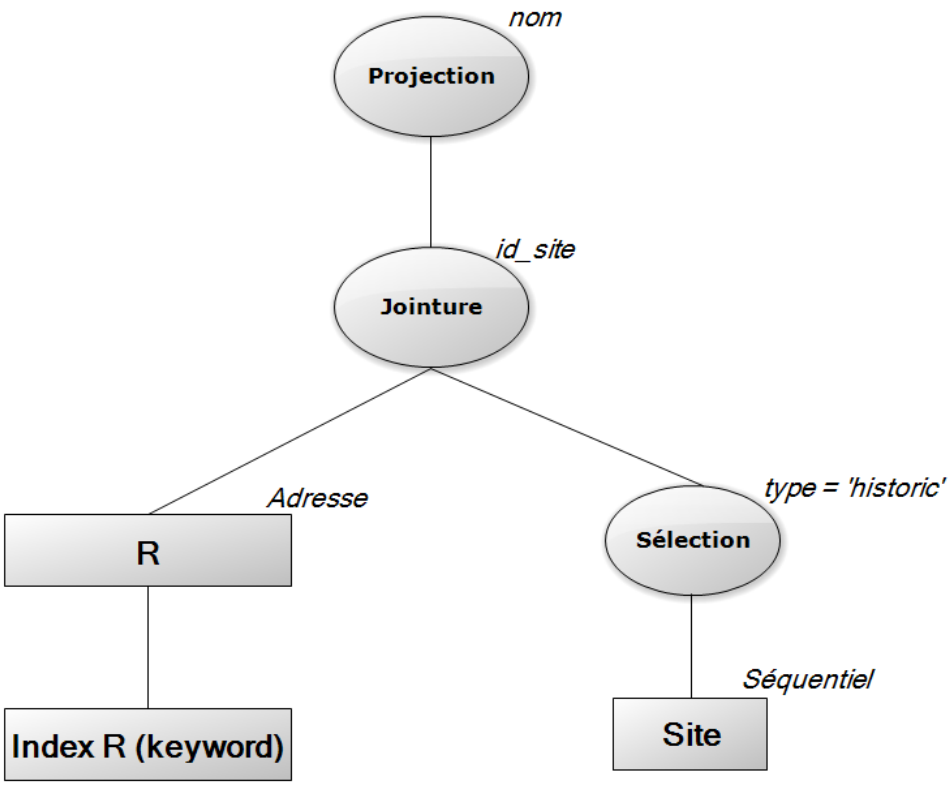

FIGURE 3 – Plan d'exécution 1

$$Cost = n \times Query_A + (n \times m) \times Query_B \tag{1}$$

En supposant qu'une des deux requêtes ait un temps d'exécution inférieur à l'autre avec un ordre de grandeur de 10^{-2}. On a donc une disjonction des cas où :

1. Iterator A ($Cost_A$) est le plus coûteux :

$$\begin{aligned} Cost_A &= n \times Query_A + (n \times m) \times Query_A \times 10^{-}2 \\ &= (n + (n \times m \times 10^{-}2)) \times Query_A \end{aligned} \tag{2}$$

   i.e. la complexité de ce cas est $\mathcal{O}(n)$

2. Iterator A ($Cost_B$) est le moins coûteux :

$$\begin{aligned} Cost_B &= n \times Query_A \times 10^{-}2 + (n \times m) \times Query_A \\ &= ((n \times 10^{-}2) + (n \times m)) \times Query_A \end{aligned} \tag{3}$$

   i.e. la compléxité de ce cas est $\mathcal{O}(n \times m)$

6.1 Plan d'exécution #1

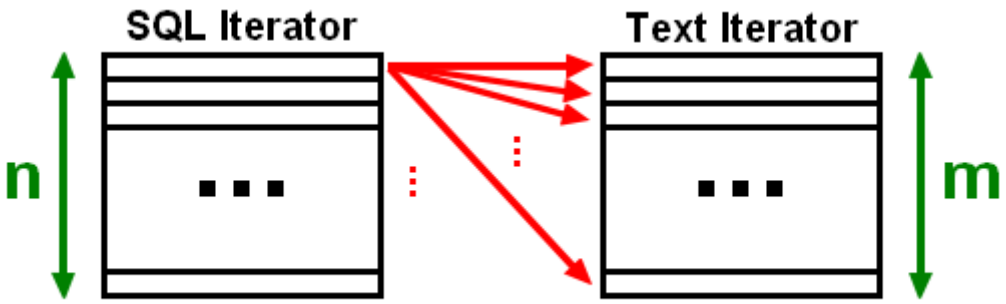

FIGURE 4 – Schéma du plan d'exécution #1

— **Avantages**

1. Rapide puisque la requête la plus lourde (SQL) n'est exécutée qu'une seule fois (équation 2).

— **Désavantages**

1. L'ordre que nous souhaitons obtenir (ordonnancement décroissant par pertinence obtenu côté requête textuelle) n'est pas conservée. En effet, c'est l'ordre de l'itérateur SQL qui est prévervée modulo les enregistrement déduits de la jointure entre les deux résultats de requêtes.

6.2 Plan d'exécution #2

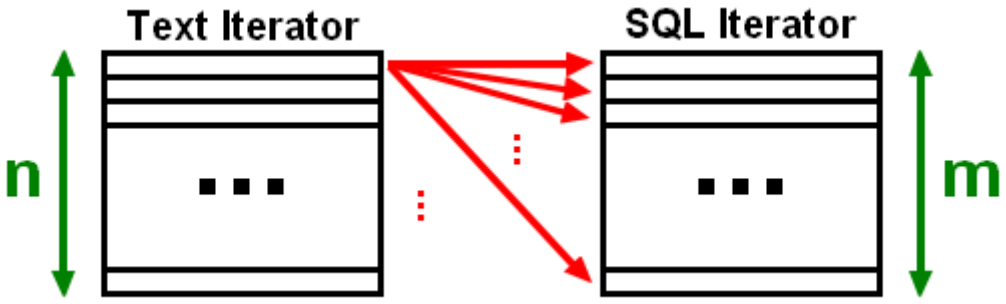

FIGURE 5 – Schéma du plan d'exécution #2

— **Avantages**

1. Conserve l'ordre du résultat produit par la requête textuelle (ordre décroissant de pertinence).

— **Désavantages**

1. L'opération de jointure est plus coûteuse car pour chaque ligne résultant de la requête textuelle, une requête SQL est effectuée (équation 3).

7 Conclusion

L'étude comparative nous amène à dire que le second plan d'exécution est plus adapté pour notre cas d'utilisation puisque nous cherchons à implémenter une recherche par mots clés (nous cherchons donc à conserver l'ordre de pertinence).

En revanche, si le critère de performance (rapidité d'exécution) est plus important que celui de la qualité du résultat (la recherche reste pertinente puisqu'elle retient les lignes obtenues par la recherche textuelle), le premier plan d'exécution peut être considéré.

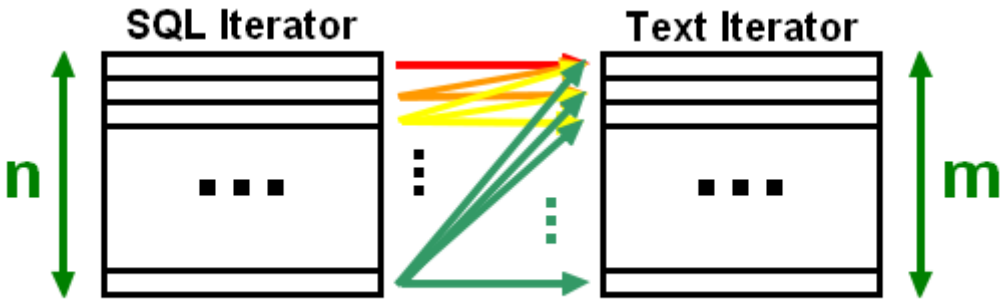

FIGURE 6 – Schéma du plan d'exécution #1 amélioré

8 Améliorations possibles

Une amélioration possible du premier plan d'exécution consisterait à trier les résultats des deux requêtes (SQL et textuelle) selon un même attribut (en l'occurrence, ici, l'attribut serait la clé de la table avec ORDER BY pour SQl et SortField FIELD_DOC pour Lucene) puis d'effectuer l'opération de jointure selon le premier plan d'exécution. Ainsi, la complexité de l'algorithme devient $\mathcal{O}(\log n)$ contre $\mathcal{O}(n)$ pour le premier plan d'exécution initial.

9 Remerciments

Notre groupe tiens à remercier Monsieur Tianxiao LIU et Monsieur Dan VODISLAV pour nous avoir confier ce projet. Au cours de cette semaine, nous avons eu l'opportunité d'approfondir nos connaissances et nos compétences en JAVA et en base de données. Sortir du cadre théorique et avoir l'occasion de travailler en équipe sur un projet d'une telle envergure nous a fait comprendre les responsabilités qu'impliquait la gestion de projet, et nous a fait gagner en maturité.